同期型言語を用いたソフトウェア検証：クルーズコントロールシステムにおける事例紹介

足立 正和　富永 孝　佐野 範佳　潮 俊光

1　はじめに

ソフトウェアの品質や信頼性を向上させる有力な方法として形式手法が近年注目されている．特に，航空機や原子力プラントのようなセーフティクリティカルなシステムにおいては，非常に高い信頼性が要求されるため，数学的枠組みに基づいた厳密な手法が必要となる．既に，ハードウェアの分野では SPIN，SMV といった種々のツールの開発により，モデル検査を用いた検証が大きな成果を挙げており，ソフトウェア開発においても形式手法の導入が期待されている．

また，航空機・自動車などの物理的なシステムの制御を行う組込みシステムを設計する際，図 1 に示すようなモデルベース開発が主流となりつつあり，ツールとしては MATLAB/Simulink が代表的なものである．モデルベース開発プロセスにおいては，開発工程の上位から下位まで一貫したモデルを利用するが，各プロセスにおいてテストパターンを用いたシミュレーションベースの検証が行われ，実装後に最終的な動作テストが行われる．よって，動作テスト時に不具合が発生した場合は上位の工程まで戻り，仕様を再検討した後，同じプロセスを繰り返す必要がある．

このような問題に対する 1 つの有効な手法として，

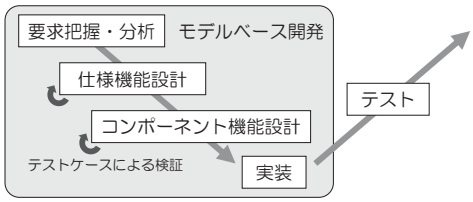

図 1　モデルベース開発

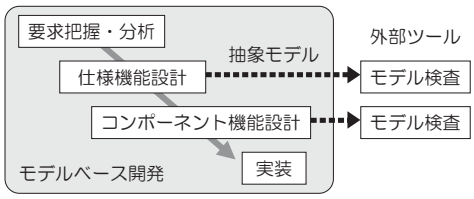

図 2　形式検証とモデルベース開発の併用

モデルベース開発と形式検証を併用し，開発段階で不具合を網羅的に取り除く方法がある．このような開発プロセスの概念図を図 2 に示す．　先述の SPIN, SMV の他にもリアルタイムシステムの検証ツールとして，Kronos，Uppaal 等が有名であり，図 2 におけるモデル検査はこれらのツールが用いられることが多い．しかし，これらのツールは検証に対しては有効であるが，対象システムをなんらかの数学モデルで記述しなければならないという制約がある．例えば Uppaal の場合，検証対象が時間オートマトンに制約されるため，ソフトウェアをなんらかの形で時間オートマトンに抽象化する必要が生じる．さらに一般的に，ソフトウェアは無限状態を持つ状態遷移系であるため，モデリング自体が困難な状況も多く存在する．

よって，モデルベース開発のプロセスに形式手法を

Software Verification by Synchronous Language: A Case Study of Cruise Control Systems.

Masakazu ADACHI, Toshimitsu USHIO, 大阪大学 大学院基礎工学研究科, Graduate School of Enginering Science, Osaka University.

Takashi TOMINAGA, Noriyoshi SANO, 豊田中央研究所, Toyota Central R&D Labs., Inc..

コンピュータソフトウェア, Vol.23, No.3 ,
[フォーラム]

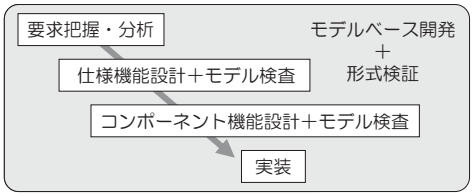

図 3　形式検証・モデルベースによる統合的開発

導入するためには，図 3 のようなモデリングから検証までを統一的に扱うことのできる枠組みが必要となる．そこで本稿では，モデルベース開発と形式検証を統合的に用いてソフトウェアを開発する枠組みの 1 つとして，厳密なセマンティクスを持つデータフロー記述言語である同期型言語 [1] に着目した．同期型言語はリアクティブシステムの検証を目的として研究され，これまでに航空機，鉄道，原子力プラント等のセーフティクリティカルなシステムの検証に多くの成果を挙げており，車載システムへの適用も報告されている [11]．同期型言語を用いることにより，MATLAB/Simulink のようなデータフローレベルでのモデリングから，モデル検査による仕様検証までを一貫して行うことが可能となる．

本稿ではクルーズコントロールシステムを例に，同期型言語による組込みシステム開発の一例を紹介する．まず，今回用いた同期型言語である Lustre について簡単に説明する．次に，組込みシステム統合開発ツールである SCADE [10] によるクルーズコントロールシステムのモデリングを簡単に紹介し，いくつかのセーフティプロパティについての検証結果を示す．最後に結論と今後の課題について簡単に述べる．

2　同期型言語による形式検証

2.1　Lustre 言語

まず同期型言語 Lustre [7] について簡単に述べる．同期型言語では一般に，入出力の関係を記述し，任意の変数や式 (*flow* と呼ばれる) は時間の関数として扱われる．さらに，全ての入力，計算，出力は同期しており 1 サイクルで実行される．例えば，Lustre において `Z = X + Y` という方程式は “全ての時間サイクル k において $\mathtt{Z}_k = \mathtt{X}_k + \mathtt{Y}_k$” という意味を持つ．方

```
node COUNTER (Reset:  bool)
return(Count:  int);
let
  Count = 0 -> if Reset then 0
               else pre(Count) + 1;
tel
```

図 4　リセットつきカウンターの Lustre 記述

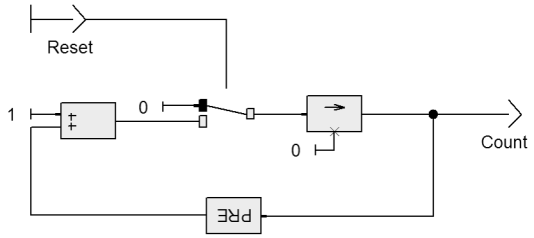

図 5　SCADE 上での `node COUNTER`

程式の表現には，変数，定数，一般算術演算，論理演算の他，逐次的な関数の処理を表す特殊な演算子 *previous* (`pre`) と *followed-by* (`->`) が定義されている．*previous* は単位時間の遅れ演算であり，*followed-by* は *flow* の初期化を行う．

- $(\mathtt{pre\ X})_k = \mathtt{X}_{k-1}$，$(\mathtt{pre\ X})_1$ は未定義．
- $(\mathtt{X\ ->\ Y})_k = \mathtt{Y}_k$，但し $(\mathtt{X\ ->\ Y})_1 = \mathtt{X}_1$

例えば，リセットつきカウンタを Lustre で記述すると，図 4 のように表現される．カウンタは初期値 0 からスタートし，前回の値に 1 を加えながら増加していく．そして，リセットが真となったときカウンタの値は 0 に更新される．

2.2　SCADE による仕様検証

本稿では，Lustre をベースとしたモデルベース組込みシステム開発環境である SCADE (Safety Critical Application Development Environment) を用いて同期型言語によるソフトウェア開発を行った．SCADE では，データフローダイアグラムとステートマシンを GUI によって図的に構築することができるため，Lustre 言語で直接プログラムすることなく，システムをモデリングすることが可能である．図 4 のカウンタを SCADE 上で構成したものが図 5 である．要

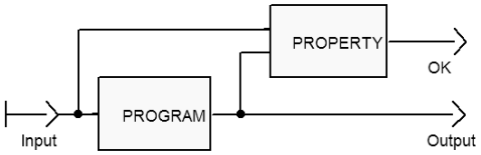

図 6 検証プログラム

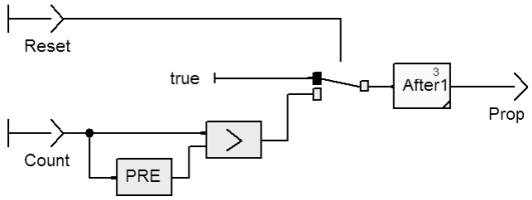

図 7 SCADE によるカウンタの仕様記述

求仕様は，モデル検査で一般的に用いられる CTL や LTL 等の時相論理式ではなく，図 6 のように，仕様自体も Lustre 言語によって記述する． つまり，`node PROGRAM` の入出力が常に `node PROPERTY` を真にするか（仕様を満たすか）どうかによって，セーフティプロパティを検証する．例えば，カウンタの例において，“$k > 1$ において，リセットが偽ならば $\texttt{Count}_k$ は常に $\texttt{Count}_{k-1}$ より大きい”という仕様を検証する場合，図 7 のような仕様記述用の node を作成し，検証プログラムを作成する．

2.3 他ツールとの比較

本稿では，組込みソフトウェア開発の 1 つの枠組みとして同期型言語を採用したが，ソフトウェア開発における形式検証の導入の方向は他にも存在する．いずれの枠組みも，対象とするシステムの性質・抽象度に合わせて選択する必要があるため，一様に議論することはできないが，前述の Kronos，Uppaal 等は同期型言語と違い “稠密な”時間を扱える点において同期型言語とは本質的に異なる．例えば，通信ネットワークのようにプロトコルが明確で時間オートマトンによるモデリングが効率的に行える場合，時間オートマトンを用いた仕様検証は多くの実用例が報告されていることからも有効な手段である．

他にも，パワートレーン系の車載システム等においては，非線形な動特性が陽に存在するため，これらを同期型言語で記述する際にはダイナミクスの離散近似が必要となる．しかし，強い非線形性を持つ場合，必ずしも離散近似によって特性を記述しきれるとは限らない．さらに，非同期な事象駆動型の割り込み処理等も同期型言語の枠組みでは扱うことができない．このような複雑な動特性を背景に持つ組込みシステムの検証には，ハイブリッドシステムの検証が 1 つの方針として挙げられる．ツールとしては HyTech [8]，CheckMate [3]，PHAVer [5] 等がこれまでに開発されており，ハイブリッドシステムの検証を実システムに適用した例としては [12] [4] 等がある．但し，ツールごとに扱えるダイナミクスのクラスが異なるため，必要に応じて使い分ける必要があり，理論的にも未整備な領域が数多く残されている．

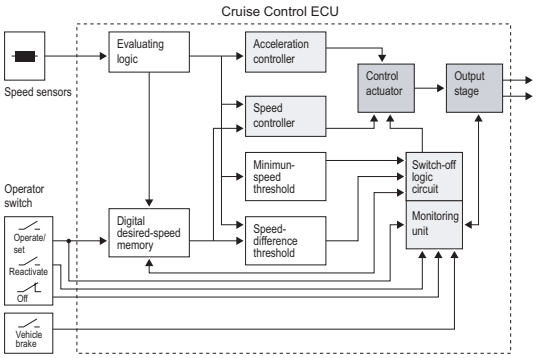

図 8 クルーズコントロールシステム [6]

3 クルーズコントロールシステムのモデル

本節ではクルーズコントロールシステムのモデリングの概要について述べる．クルーズコントロールシステムとは，運転者が所望の車速をセットすると，アクセルペダル操作をしなくても記憶車速で定速走行するシステムであり，高速道路等における長時間運転の疲労を軽減することができる．同時に，緊急時の作動解除の確実性や，機器の故障に対するフェイルセーフのような高い安全性が要求される [15]．クルーズコントロールシステムの概略図を図 8 に示す． クルーズコントロールシステムは，電源の ON・OFF や車速のセットを制御するコントロールスイッチと，

車速等のセンサ情報をもとにアクチュエータを制御する．システム全体は，(1) 外部情報から現在の状態を取得する前処理，(2) 制御モードを決めるコントローラ，(3) 最終的な出力を計算する後処理という 3 つのサブシステムを基本構造として持つ．以上を各種仕様書 [9][14] をもとに SCADE を用いてモデリングした．前処理（Evaluating logic, Digital desired-speed memory, Minimum-speed threshold, Speed-difference threshold）は主にスイッチの ON・OFF などのエッジの検出，閾値との比較演算等から構成され，後処理（Control actuator, Output stage）はコントローラの出力（制御モード）に応じた代数演算がメインとなる．よって，前処理・後処理に関しては，主にデータフローによって記述を行った．一方コントローラ部（Acceleration controller, Speed controller, Switch-off logic circuit, Monitoring unit）は，前処理された情報をもとに制御モードを適切に切り替え，アクチュエータの制御則を決めるため，主にステートマシンを用いて記述されている．得られたシステムの詳細は省略するが，全体としては 18 個の入力（15 個のブール型変数と 3 個の実数型変数）を持つシステムとなった[†1]．前処理，コントローラ，後処理はそれぞれ階層的に構成されており，全体として 18 個の node からなっている．また，ステートマシン（コントローラ）は 4 つの状態を持ち，前処理によって生成された 8 個のイベントの組み合わせにより，状態遷移が定義されている．システム全体を Lustre 言語で記述した場合，約 500 行のプログラムとなる．航空機における適用例ではフラップの制御部で 1905 行，センサシステムで 3454 行，フライトコントロールシステムで 4267 行程度と報告されており [2]，クルーズコントロールシステムは中規模サイズの問題であると言える．

4 セーフティプロパティと検証結果

前節で得られたモデルに対して，セーフティプロパティの検証を行った．SCADE では他のツール同様に検証性質が満たされない場合は反例を示す．反例が出た場合は，それをトレースすることにより効率的にデバッグが行える．また，検証のタイプとして prove と debug があり，どちらも帰納的な深さ探索と SAT を組み合わせた手法によりセーフティプロパティの検証を行う [13]．prove が通常の検証であるのに対して，debug はパスの深さ *depth* をパラメータとして持ち，長さ *depth* 以内の反例を探索する場合に用いられる．検証性質が prove ではチェックできなかった場合には，debug によってある程度の深さで反例を探し，対応する箇所を修正した後 prove でもう一度検証するというサイクルが SCADE における検証の一般的な手順である．本稿では，クルーズコントロールシステムの取扱説明書 [9] をもとに安全性にかかわる項目を抜粋し，18 の検証性質を設定した．以下にいくつかの検証項目とその結果についてまとめる．

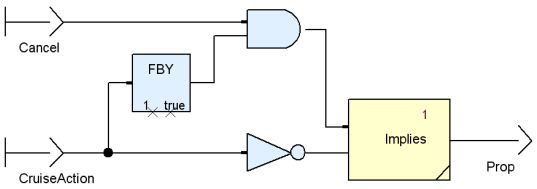

図 9　制御解除の仕様

制御解除：クルーズコントロールシステムでは，何らかの制御解除条件（システム異常，悪天候によるセンサ不良等）が成立した場合には，定速走行を解除し通常の運転に戻る．今回は 4 つの制御解除に関わる条件に対して検証を行った．いずれの場合も，“制御中に制御解除条件が発生したら，制御解除される”
$(\texttt{pre}(\texttt{CruiseAction}) \wedge \texttt{Cancel}) \Rightarrow \neg\texttt{CruiseAction}$
と記述でき，仕様記述ブロックは図 9 のようになる．図中のブロック `FBY(CruiseAction,1,true)` は `true -> pre(CruiseAction)` と等価である．この仕様に対しては，いずれの場合も 1 秒未満で valid（検証性質が満たされている）という結果を得た．

セット車速上限超走行時における制御開始：クルーズコントロールによる制御を開始する場合，現車速がセット車速として記憶され，セット車速で定速走行を行うようになっている．ここで，制御を開始する際にクルーズコントロールの制御範囲外の速度で走行していた場合，セット車速は制御範囲の上限値としてセッ

†1 詳細なモデルは，http://ushiolab.sys.es.osaka-u.ac.jp/ut/scade/acc.html を参照．

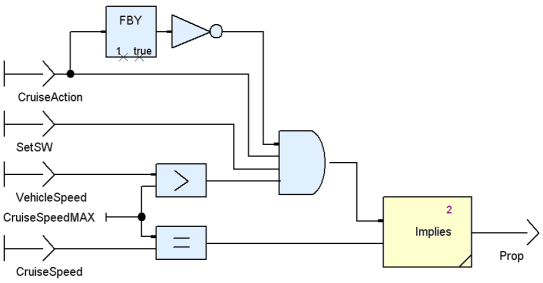

図 10 セット車速上限を超えた制御開始の仕様

トされなければならない．仕様としては，“セットスイッチが押されて制御中状態になる時に現車速が上限値を上回っている時，セット車速はセット上限値に設定される”，つまり，

$$(\texttt{pre}(\neg\texttt{CruiseAction}) \wedge \texttt{CruiseAction} \wedge \texttt{SetSW} \wedge (\texttt{VehicleSpeed} > \texttt{CruiseSpeedMAX})) \Rightarrow (\texttt{CruiseSpeed} = \texttt{CruiseSpeedMAX})$$

となり，仕様記述ブロックは図 10 のようになる．この段階では，前述の検証結果から制御解除は常に正しく動作することが証明済みであるため，フェールなどの例外的な制御解除条件は無視できる．そこで，より効率的な探索を行うため assertion 記述により変数に制約を設け，制御解除は決して発生しないという条件の下 prove ストラテジーによる検証を行ったところ，2 秒未満で valid という結果を得た．

アクセル操作による加速：定速走行中でも，前方車の追い越す必要がある場合などは，運転手によるアクセル操作が優先され，常にアクセルによる加速が可能である．もし，運転手によるアクセル操作よりクルーズコントロールが優先されるような事があれば，咄嗟の加速による衝突回避などが行えず事故につながる恐れがあるため，重要な検証項目の 1 つである．フェール等の発生はないものとして，prove ストラテジーによる検証を行ったところ、2 秒未満で反例が見つかり，確認したところアクセルペダルと同時にブレーキペダルが踏まれると，ブレーキペダルの方が優先され加速できないというシナリオであることが分かった．これはモデルの不備によるものであったため，対応するモデルの箇所を修正し，再び検証を行ったところ最終的に valid という結果を得た（計算時間は 1 秒未満）．

以上でいくつかの検証項目を紹介したが，全ての検証項目についてまとめたものが表 1 である．18 の検証項目の内 12 については valid を導くことができた．その中で 3 つの項目に関しては，反例をもとにモデルの不備を修正した後 valid となった．残りの 6 つに関しては，現段階では回答を得ることができなかった．それらの特徴としては，(1) より詳細なモデルが必要であるもの（5 項目）と，(2) 連続変数が陽に表れる検証性質 (1 項目) の 2 つが挙げられる．(1) に関しては，今回行ったモデリングでは記述していないようなレベルの振る舞いが仕様として記述されているために，検証性質そのものが記述できなかったケースである．(2) に関しては，クルーズコントロールシステムのような連続変数が多く存在するシステムでは，割り算，非線形演算などが存在し，モデル検査を行った際に計算が収束しなくなってしまうケースである．Design verifier では加算・減算・乗算（ただし乗算に関しては一方が定数）以外の演算は，シンボリックに操作できないため，indeterminate を回答する．モデル内に非線形演算などの “可解でない”演算が入っている場合でも，検証性質によってはそれらの部分を無視でき，検証が可能であるが，そうでない場合は困難である．よって，SCADE で整数や実数が多く存在するシステムを対象とする場合には前述の限界に留意しながらモデリングを行う必要があり，非線形演算などは必要応じてマッピング等で置き換えることが必要である．しかし，これらの作業はモデルの厳密性を失う可能性があり，また経験を必要とする側面でもある．

5 まとめ

本稿ではクルーズコントロールシステムを例に挙げ，組込みソフトウェア開発に形式検証を適用した事例を紹介した．同期型言語 Lustre をベースにした組み込みシステム統合開発環境 SCADE を用いて，実際にセーフティプロパティの検証を行いその有効性を示した．しかし同時に，ソフトウェア開発に形式手法を導入する際の課題も明らかになったと言える．特

表 1　検証結果

検証の可否	検証工程の説明	項目数
可能	モデルの修正なく valid	9
	反例をもとにモデルを修正した後 valid	3
不可能	検証性質を記述するには，より詳細なモデルが必要	5
	連続変数に起因する計算の発散	1

に，今回のような自動車分野においてのソフトウェア開発は，物理的な環境が外在するという点において問題が顕著に現れており，クルーズコントロールシステムの場合には連続変数の複雑な演算をいかに扱うかが最大の問題となった．いずれにせよ，実際にソフトウェア開発に形式手法を導入する際には，ツールごとの特徴と制約を把握した上でモデリングを行わなければ効果的な結果を得ることは難しく，検証すべき性質を必要最小限に記述するモデルとツールの組み合わせをいかに選択するかが重要である．

今回得た結果をもとに，他の車載システムへの適用を試みるとともに，他ツールとの比較検討，そして自動車ドメインに特化した形式手法の枠組みの構築が今後の課題である．

参 考 文 献

[1] Benveniste, A., Caspi, P., Edwards, S. A., Halbwachs, N., Guernic, P. L., and de Simone, R.: The synchronous languages 12 years later, *Proceedings of the IEEE*, Vol. 91,No. 1(2003), pp. 64–83.

[2] Bouali, A. and Dion, B.: Formal verification for model-based development, *SAE World Congress*, Detroit, MI, April 2005, pp. 05AE–235.

[3] Chutinan, A.: *Hybrid system verification using discrete model approximation*, Ph.D. dissertation, Department of Electrical and Computer Engineering, Carnegie Mellon University, 1999.

[4] Fehnker, A. and Krogh, B. H.: Hybrid systems verification is not a sinecure: the electronic throttle control case study, *Automated Technology for Verification and Analysis 2004*(Wang, F.(ed.)), LNCS 3299, Springer-Verlag, Berlin, 2004, pp. 263–277.

[5] Frehse, G.: PHAVer: Algorithmic verification of hybrid systems past HyTech, *Hybrid Systems: Computation and Control*(Morari, M. and Thiele, L.(eds.)), LNCS 3414, Springer-Verlag, march 2005, pp. 258–273.

[6] Robert Bosch GmbH.: *Automotive Electrics and Electronics*, Society of Automotive Engineers, 1999.

[7] Halbwachs, N., Caspi, P., Raymond, P., and Pilaud, D.: The synchronous dataflow programming language Lustre, *Proceedings of the IEEE*, Vol. 79,No. 9(1991), pp. 1305–1320.

[8] Henzinger, T. A., Horowitz, B., Majumdar, R., and Wong-Toi, H.: Beyond HyTech: Hybrid systems analysis using interval numerical methods, *Hybrid Systems: Computation and Control*(Krogh, B. H. and Lynch, N.(eds.)), LNCS 1790, Springer-Verlag, 2000, pp. 130–144.

[9] http://gazoo.com/auto/J/w_torisetsu_top.asp.

[10] http://www.esterel-technologies.com/

[11] Köhler, A. and Kant, D.: Use of formal verification for the software development in the automotive area, http://www.esterel-technologies.com/technology/WhitePapers/overview.html.

[12] Stursberg, O., Fehnker, A., Han, Z., and Krogh, B. H.: Verification of a cruise control system using counterexample-guided search, *Control Engineer Practice*, Vol. 12,No. 10(2004), pp. 1269–1278.

[13] Sheeran, M., Singh, S., and Stålmarck, G.: Checking safety properties using induction and a SAT-solver, *Proceedings of the 3rd International Conference on Formal Methods in Computer-Aided Design*, LNCS 1954, Springer-Verlag, London, UK, 2000, pp. 108–125.

[14] トヨタ自動車株式会社: クラウン マジェスタ配線図集, 2004.

[15] 荒井宏: 自動車の電子システム, 理工学社, 1992.